C Tutorial and Review Session:
Dewen, Weiwen, Priyankar

Topic 1: Input and Output (Dewen)
Input: scanf(), from your keyboard. (not so important for this class)
Output: printf(), to the screen.

```c
#include <stdio.h> //you must include this file
int main()
{
   int a;
   printf("please type your input.\n");
   // &a is the address of variable a
   scanf("%d", &a);
   printf("variable a = %d\n", a);
}
```

You can use these expressions to format your variable.

%d	Input or display as decimal
%x	Input or display as hex
%c	Input or display as character
%s	Input or display as string
%f	Input or display as floating point
%p	display as a pointer (64-bit hex)

You can input/output two or more variables at once.

```c
#include <stdio.h>
int main()
{
   int a, b;
   printf("please type your input.\n");
   // &a is the address of variable a
   scanf("%d,%d", &a,&b);
   printf("variable a = %d, b = %d\n", a, b);
}
```

You can also use **getchar()** and **putchar()** functions
The **int getchar(void)** function reads only a single character from the keyboard and returns an integer.

The **int putchar(int c)** function puts the passed character on the screen and return the same character. One character a time.

```c
#include <stdio.h>
int main() {
  int c;
  printf("Enter a value :");
  c = getchar();
  printf("\nYou entered: ");
  putchar(c);
  return 0;
}
```

Overall, **scanf() and printf()** are more general and useful.

Topic 2: Types and Operators (Dewen)

2.1 Types

There are a few basic types:

- Integers: can be either positive or negative.
  char (1 byte), int (4 bytes), short (2 bytes), long (8 bytes).
- Unsigned Integers. Can only be positive. unsigned char, unsigned int, unsigned short ...
- Floating point numbers. float, double
- Structures (not covered)

Q: How to test how many bytes each variable type has?

```
printf("char type has %d bytes\n", sizeof(char));
printf("int type has %d bytes\n", sizeof(int));
printf("short type has %d bytes\n", sizeof(short));
printf("long type has %d bytes\n", sizeof(long));
```

Q: How to assign a decimal, hex, binary number to a variable?

```
int x = 3;
printf("x = %d\n",x);
x = 0x10 or 0X10;
printf("x = %d\n",x);
x = 0b0011;
printf("x = %d\n",x);
```

2.2 Arithmetic Operators

Operator	Meaning of Operator
+	addition
-	subtraction
*	multiplication
/	division
%	remainder

Example:

```
int main()
{
   int a = 9, b = 4, c;
   c = a+b;
   printf("a+b = %d \n",c);
   c = a-b;
   printf("a-b = %d \n",c);
   c = a*b;
   printf("a*b = %d \n",c);
   c = a/b;
```

```
   printf("a/b = %d \n",c);
   c = a%b;
   printf("Remainder when a divided by b = %d \n",c);
}
```

2.3 Assignment Operators

An assignment Operator is used for assigning a value to a variable. The most common one is **=.**

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

```
// simple assignment of a constant
a = 7;
// simple assignment involving more than 1 variable such as a = b + c
a = 1; b = 1; c = 1;
a = b + c;
// updating a single variable such as
a = a + 1;
// shortcut
a += 1; <==> a = a + 1;
```

2.4 Increment and Decrement Operators

- Increment: ++, increase the value by 1.
- Decrement: --, decrease the value by 1.

Example:

```
int main(){
   int a = 10, b = 100;
   printf("++a = %d \n", ++a);
   printf("--b = %d \n", --b);
}
```

There is a big difference between a++ and ++a:

- ++a: return a+1
- a++: return a

After each operation a = a + 1

Example:

```c
int main(){
   int a = 10, b = 100;
   printf("++a = %d \n", ++a);
   printf("--b = %d \n", --b);
   printf("a++ = %d \n", a++);
   printf("b++ = %d \n", b++);
   printf("a = %d \n", a);
   printf("b = %d \n", b);
}
```

The result:

```
++a = 11
--b = 99
a++ = 11
b++ = 99
a = 12
b = 100
```

2.5 Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

2.6 Logic Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results are true or false.

Operator	Meaning
&&	Logical AND. True only if all operands are true
||	Logical OR. True if either one operand is true
!	Logical NOT. True only if the operand is 0

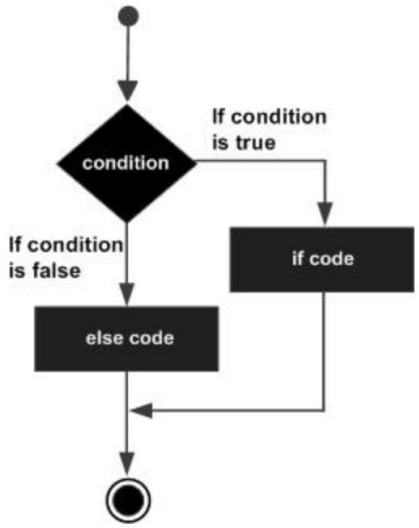
Topic 3: Conditions (Dewen)

3.1 if statement.

It allows us the check if an expression is true or false, and execute different code according to the condition.

Syntax

```
if(condition1){
   do something;
}
else if(condition2){
   do something;
}
else{
   do something;
}
```



In C, we use 0 to represent false, none 0 to represent true. An evaluate expression would return 0 if it's false, 1 if it's true. Note that C will consider any non-zero value to be true as a condition for an if statement.

```
int x = 10;
if(x){
    printf("x is true\n");
}
else{
    printf("x is false\n");
}
int y = 0;
if(y){
    printf("y is true\n");
}
else{
    printf("y is false\n");
}
```

```
printf("condition x > 1 is %d\n", x>1);
printf("condition x < 1 is %d\n", x<1);
```

You can use relational operation (“==”, “<”, “>”, “!”, etc.) to evaluate an expression. If you want to evaluate two or more than two expressions at the same time, use “&&” and “||”.

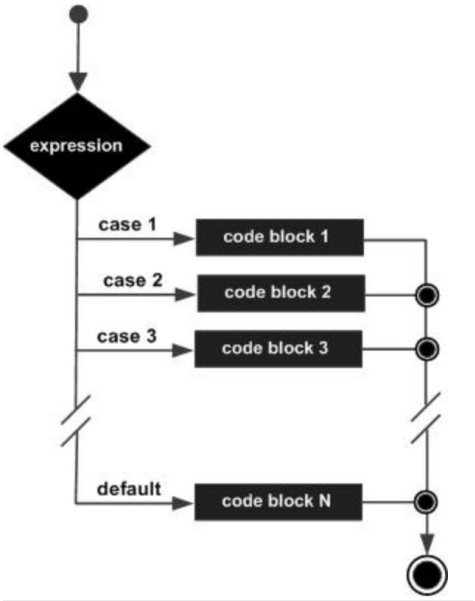
```
// use not operation
int target = 9;
if(target != 10){
   printf("Target is not equal to 10.\n");
}
// use && and ||
int x = 1;
int y = 2;
int z = 3;
if (x < y && z > y){
   printf("x is smaller than y and y is smaller than z\n");
}
if(x < y || z < y){
   printf("x is smaller than y or z is smaller than y\n");
}
```

3.2 switch statement.

A variable/expression can have different values, each value is a case, we hope to do different things in different cases.

Syntax:

```
switch(expression){
   case constant-expression:
      statement(s);
      break;
   case constant-expression:
      statement(s);
      break;
   default:
      statement(s);
}
```



Q: Suppose a student's grade can be "A", "B", "C" or "D", we want to give different comments based on his/her grade.

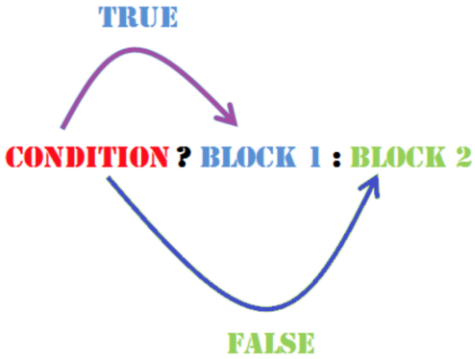
```
// Of course we can use if else statement
char grade = 'A';
if(grade=='A'){
   printf("Excellent!\n");
}
else if(grade=='B'){
   printf("Good!\n");
}
else if(grade=='C'){
   printf("Em-em...\n");
}
else if(grade=='D'){
   printf("Try harder!\n");
}
else{
   printf("Wrong character!");
}
// Using switch case would be more straightforward.
switch(grade){
case 'A':
   printf("Excellent!\n");
   break;
case 'B':
   printf("Good!\n");
   break;
case 'C':
   printf("Em-em...\n");
   break;
```

```
case 'D':
   printf("Try harder!\n");
   break;
default:
   printf("Wrong character!");
   break;
}
```

3.3 conditional operator. *(not required)*

Syntax:

```
expression1 ? expression2 : expression3
```




```
int num = 10;
(num > 5)?printf("num > 5\n"):printf("num <= 5\n");
```

Sometimes it can be used to replace if else.

Quiz:

Q3-1: Input three integers: x, y, z, and print them in ascending order.

```
input: x, y, z = 8, 10, 5;
output: x, y, z = 5, 8, 10;
```

Q3-2: Use conditional operator to do this: if grade >= 90 print A, if grade is 60 to 90 print B, if grade is less than 60 print C.

```
input: grade = 50
output: C
```

Topic 4: Loops (Weiwen)

Relieving programmers' labor is one of the most important things in a programming language. Loops can help programmers to avoid rewriting the same code block many times. Instead, if the code block must be repeated, it only needs to be written once.

In the following text, we are going to talk about one of the most commonly used loops: the while loop. It supplies the ability to run a code block multiple times.

Consider the task of the summation of numbers from 1 to 5 and storing it to a variable ***x***. The straightforward way is to directly write the statement ***"x=1+2+3+4+5;"***. However, if our task becomes the summation of numbers from 1 to 1,000 or from 1 to 100,000, how do you deal with these tasks? It is obvious that we cannot write down statements by hand. In the following text, let's take the summation of numbers from 1 to 5 as an example to introduce the while loop in C.

4.1 The syntax and data flow of "while loop".

The syntax and structure of the while loop are illustrated as follows:

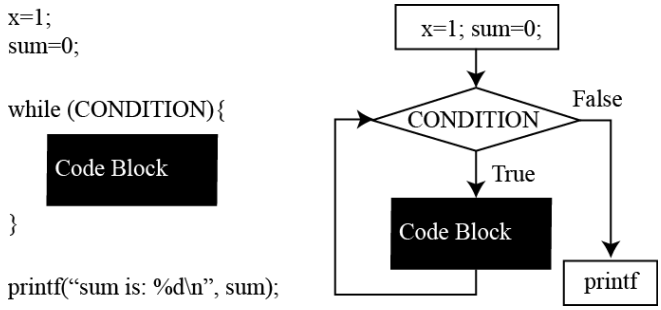


Figure 4.1: Illustration of while loop: left-hand, while loop C syntax; right-hand: dataflow of while loop

The while loop contains two parts: (1) a condition, (2) code block.

As shown in the right-hand figure in Fig. 4.1, if the condition is True, the while loop will enter the loop body and then execute the "Code Block". At the end of the code block, the program will automatically go back to the head of the *while loop*, and check the condition again. Such a procedure will iteratively conducted until the condition becomes False.

4.2 Compute x=1+2+3+4+5 using while loop.

To conduct a summation operation, we use a "sum" variable to hold the temporary results, which is initialized as 0 at the beginning. In addition, we create an "x" variable to traverse the numbers from 1 to 5, which is initialized as 1.

(1) Design of code block:

We call an execution of the code block as one iteration. In each iteration, we will do the following two things: *i)* add the value of x to temporary variable “sum”; *ii)* increase the value of x by 1.

In consequence, we have the following code snippet.

```
// x=1+2+3+4+5;
int x = 1;
int sum = 0;

while (CONDITION){
   sum = sum + x;
   x = x + 1;
}

printf("Sum is: %d\n",sum);
```

(2) Design of condition:
So far, we have implemented the code block to be executed iteratively. The next problem is when the loop should be terminated, and how to terminate the loop.

In our example, the loop needs to be terminated once the value of ***x*** *is larger than* 5 (WHEN). And we need to *change the CONDITION to be False* once the value of ***x*** *exceeds* 5, while we need to *keep the CONDITION to be True* if the value of ***x*** *is less than or equal to* 5 (HOW). Based on the above analysis, one possible CONDITION can be ***“x<=5”***.

Now, we can write the full C program using the while loop to calculate the summation of numbers from 1 to 5.

```
// x=1+2+3+4+5;
#include <stdio.h>

int main(){
     int x = 1;
     int sum = 0;

     while (x <= 5){
        sum = sum + x;
        x = x + 1;
     }

     printf("Sum is: %d\n",sum);
     return 0;
}
```

Quiz:

Q4-1: Using the while loop to conduct the following calculation: 1+2+3+...+100.
Q4-2: Using the while loop to conduct the following calculation: 1+2+3+...+1000.
Q4-3: Using the while loop to conduct the following calculation: 1+3+5+7+...+99.
Q4-4: Using the while loop to conduct the following calculation: (-100)+(-99)+...+(-1).

4.3 Extension:

In addition to the “while loop”, the “for loop” is also commonly used in C program. “For loop” not only contains the CONDITION, but also have an initialization (INIT), and finalization (FIN), whose syntax is listed as follows.

```
for(INIT; CONDITION; FIN){
   // Code Block
}
```

INIT is to initialize the iteration variable. FIN is to conduct a common operation after the execution of the code block.

In the while loop example, we initialize ***x*** before the while loop by ***“x=1;”***, and we perform the increment operation ***“x=x+1;”*** at the end of Code Block. Usually, these two operations will be added into “for loop” as INIT and FIN, respectively.

Now, we can use “for loop” to re-implement the function of “x=1+2+3+4+5;” as follows.

```
// x=1+2+3+4+5;
#include <stdio.h>

int main(){
     int sum = 0;
     int x;
     for (x = 1; x <= 5; x=x+1){
        sum = sum + x;
     }

     printf("Sum is: %d\n",sum);
     return 0;
}
```

The program is more concise!

Topic 5: Function (Weiwen)

The C Function can further relieve programmers' labor by assembling a group of statements that work together in a *function*. As such, we can use one statement (function call) to replace the group of statements once the function is defined.

Every C program has at least one function: the main function, ***main()***. This is the entrance of a C program. A function is a group of statements that work together to perform a task. In addition to the main function, programmers can self-define their own functions.

After assembling a group of statements into a function, it can be reused many times. The function is an important element for building a large project. For example, to test the functional correctness of tasks, we can simply test each task, instead of testing all codes together. In addition, functions can reduce the overhead to revise the program. We can see this in the following examples.

Let's come back to the summation problem. If we want to integrate the calculations from Quiz 4-1 to 4-4 to one program, the following codes give the solution without a self-defined function.

```c
#include <stdio.h>

int main(){
     int sum = 0;
     int x;
     for (x = 1; x <= 100; x=x+1){
        sum = sum + x;
     }
     printf("Sum of 1+2+...+100 is: %d\n",sum);

     sum=0;
     for (x = 1; x <= 1000; x=x+1){
        sum = sum + x;
     }
     printf("Sum of 1+2+...+1000 is: %d\n",sum);

     sum=0;
     for (x = 1; x <= 99; x=x+2){
        sum = sum + x;
     }
     printf("Sum of 1+3+...+99 is: %d\n",sum);

     sum=0;
     for (x = -100; x <= -1; x=x+1){
        sum = sum + x;
     }
```

```
    printf("Sum of (-100)+(-99)+...+(-1) is: %d\n",sum);

    return 0;
}
```

The above codes are tedious!

We make an observation that all calculations in Quiz 4-1 to Quiz 4-4 perform the same task: summation of an **"**Arithmetic Sequence**"**. As such, we can create a ***ArithSeq*** function for the same task with different inputs.

5.1 What does a function look like?

First, each function has its own name. For example, the entrance function of C program has the name of "main". And for the summation task, we have the function name "ArithSeq". Unlike "main", "ArithSeq" is a user-defined name, which can be modified, such as "AS", "AriS", "ASeq", etc., but cannot be the type name that has already been used, such as "int", "float", etc.

Each task has different inputs, resulting in different outputs, like Quiz 4-1 to Quiz 4-4. Correspondingly, each function has inputs (called arguments) and output (return value).

The syntax of a function is given as follows:

```
int ArithSeq(int start, int end, int diff){
    int sum=0;

    // Function Body

    return sum;
}
```

For the previous summation task, we need three inputs: the start number, the end number, and the common difference, which are defined as three arguments (start, end, diff) in the above "ArithSeq" function. In the function, we define a variable sum to store the result, which is finally returned as output.

Now, we can use the loop to realize the ArithSeq function as follows.

```
int ArithSeq(int start, int end, int diff){
    int sum=0;
    int x;
    for (x = start; x <= end; x=x+diff){
```

```
        sum = sum + x;
    }
    return sum;
}
```

5.2 How to invoke a function?

After creating a user-defined function, we can modify the main function to utilize (invoke) ArithSeq for a more concise program. It is listed as follows.

```
#include <stdio.h>

int ArithSeq(int start, int end, int diff){
    int sum=0;
    int x;
    for (x = start; x <= end; x=x+diff){
        sum = sum + x;
    }
    return sum;
}

int main(){
    int sum=0;
    sum = ArithSeq(1,100,1);
    printf("Sum of 1+2+...+100 is: %d\n",sum);

    sum = ArithSeq(1,1000,1);
    printf("Sum of 1+2+...+1000 is: %d\n",sum);

    printf("Sum of 1+3+...+99 is: %d\n",ArithSeq(1,99,2));

    sum = ArithSeq(-100,-1,1);
    printf("Sum of (-100)+(-99)+...+(-1) is: %d\n",sum);

    return 0;
}
```

In the main function, we invoke the self-defined ArithSeq function by passing arguments to it. For the example of calculating 1+2+...+100, we know that the start value is 1, the end value is 100, and the common difference is 1, as such, we invoke the function as ArithSeq(1, 100, 1).

We use statement “sum = ArithSeq(1, 100, 1);” to assign the output of ArithSeq to variable sum.

Note that since ArithSeq returns an int, we can directly invoke it and print the return value in *printf*. For example, ArithSeq(1, 99, 2) in the above codes.

Quiz:

Quiz5-1: Write a C function that uses the shift and add operations to calculate the sum of "Geometric Progression" with a common ratio of 2. Note that the start value (a0) and the number of terms (N) can be varied. Use `int` values in your program.

$$sum = a_0 + a_0 \times 2 + a_0 \times 2^2 + \cdots + a_0 \times 2^{N-1}$$

```c
#include <stdio.h>
int GP(int a0, int N)
{
   int sum= 0;
   int i = 0;
   int ai = a0;

   while (i < N) {
       sum = sum + ai;
       ai = ai << 1;
       i = i + 1;
   }
   return sum;
}
int main(){
   // Start with 1, and 5 terms in total
   printf("1+2+4+8+16 = %d\n",GP(1,5));
   // Start with 4, and 8 terms in total
   printf("4+8+16+32+64+128+256+512 = %d\n",GP(4,8));
   return 0;
}
```

Topic 6: Arrays and Pointers (Priyankar)

Arrays are special variables which can hold more than one value under the same variable name, organised with an index.

```
#include <stdio.h>
int main()
{
   //Define an array of 5 integers
   int a[5];
   //Fill the array
   a[0] = 1;
   a[1] = 2;
   a[2] = 3;
   a[3] = 4;
   a[4] = 5;
   //Print elements
   printf("The second element is: %d \n", a[1]);
return 0;
}
```

The index denotes the position in the array which starts from 0. Add the lines below in your program to test this:

```
for (int i=0; i <5; i++)n
{
   printf("The value at the position %d in the array is: %d \n", i, a[i]);
}
```

The array is stored in a memory location. Add the following line to see the location:

```
// Address of the array
printf("The address of the array is: %p \n", &a);
```

To see the address of a particular element:

```
//Address of the 3rd element
printf("The address of the 3rd element in the array is: %p \n", &a[2]);
```

View the addresses of all the array elements:

```
//Print addresses
for (int i=0; i <5; i++)
{
  printf("The address at the position %d in the array &a[%d] is: %p \n",
i, i, &a[i]);
}
```

Pointers are used for storing address of dynamically allocated arrays.

```
//Declare a pointer a pointer
int *ptr = &a;
```

See the value of the array element by pointer:

```
//Print value of array by pointer
printf("Value of the first element is %d \n", *ptr);
```

To see the value of a pointer which is basically address:

```
//Print value of a pointer
printf("Value of the pointer is %p \n", ptr);
```

Print value of an array element by pointer:

```
//Print value of 3rd element in array by pointer
printf("Value of the 3rd element is %d \n", *(ptr + 2));
```

Print address of an array element by pointer:

```
//Print address of 3rd element
printf("Address of the 3rd element is %p \n", (ptr + 2));
```

Increment operation of pointer:

```
//Pointer increment
ptr++;
//Print value of the element in array by pointer
printf( "Value of the element is %d \n", *ptr );
//Print address of the element
printf( "Address of the element is %p \n", ptr );
```

Enter the value of an array at a particular position:

```
//Enter the value of 3rd element
printf("Please enter the 3rd element in the array: \n");

scanf("%d", &a[2]);

//Print the elements in the array

for (int i=0; i <5; i++){
   printf("The value at the position %d in the array is: %d \n", i, a[i]);
}
```

Now, Enter the value of the array at a particular position:

```
//Decrease the pointer to the previous position
ptr--;
//Enter the value of the 4th element
printf("Please enter the 4th element in the array: \n");
```

```
scanf("%d", (ptr + 3));
//Print the elements in the array
for (int i=0; i <5; i++){
   printf("The value at the 3rd position in the array is: %d \n",
*(ptr+i) );
}
```

Topic 7: Strings (Priyankar)

Strings in C are arrays of characters.

```c
#include <stdio.h>
#include <string.h>

int main()
{
      //Declare a string without the size
      char car[] = "Toyota RAV4";

      /*is same as with declaring with string size*/
      //char car[11] = "Toyota RAV4";

      //Print the string
      printf("The car is: %s \n", car);

      //Print string length
      printf("String length is %lu\n",strlen(car));

      //String comparison to check whether two strings are same

      if (strncmp(car, "Toyota Camry",8) == 0) {

          printf("Same vehicle\n");

      } else {

          printf("SUV vs Sedan\n");

      }

      //String concatenation to join two strings

      char carsegment[] = " is a SUV starting from 25K";

      strncat(car,carsegment,9);
      printf("%s\n",car);

      return 0;
}
```

To run your C code on local machine in Integrated Development Editor (IDE), you can download Visual Studio Code from the following link:
https://code.visualstudio.com/Download